

# A figure of merit for describing the performance of scaling of parallelization

János Végh<sup>a</sup>, Péter Molnár<sup>b</sup>, József Vásárhelyi<sup>a</sup>

<sup>a</sup>*University of Miskolc, Hungary*

<sup>b</sup>*PhD School of Informatics, University of Debrecen, Hungary*

---

## Abstract

With the spread of multi- and many-core processors more and more typical task is to re-implement some source code written originally for a single processor to run on more than one cores. Since it is a serious investment, it is important to decide how much efforts pays off, and whether the resulting implementation has as good performability as it could be. The Amdahl's law provides some theoretical upper limits for the performance gain reachable through parallelizing the code, but it needs the detailed architectural knowledge of the program code, does not consider the housekeeping activity needed for parallelization and cannot tell how the actual stage of parallelization implementation performs. The present paper suggests a quantitative measure for that goal. This figure of merit is derived experimentally, from measured running times, and number of threads/cores. It can be used to quantify the used parallelization technology, the connection between the computing units, the acceleration technology under the given conditions, or the performance of the software team/compiler.

*Keywords:* multi-core, parallelization, performance, scaling, figure of merit

---

## 1. Introduction

The computer manufacturing technology is not any more able to produce quicker processors, see Agarwal et al. (2000). The crisis of the computing

---

*Email addresses:* J.Vegh@uni-miskolc.hu (János Végh), pmolnar@lib.unideb.hu (Péter Molnár), vajom@mazsola.iit.uni-miskolc.hu (József Vásárhelyi)

experienced since cca. year 2000, see Fuller and Millett (2011), increased the demand towards parallel computing.

On hardware side: "*Processor and network architectures are making rapid progress with more and more cores being integrated into single processors and more and more machines getting connected with increasing bandwidth. Processors become heterogeneous and reconfigurable.*", see S(o)OS project (2010). On software side: "*parallel programs ... are notoriously difficult to write, test, analyze, debug, and verify, much more so than the sequential versions*", see Yang et al. (2014). In addition, the typical real-life programs show complex parallelization behavior, see Yavits et al. (2014), and also the apparently massively parallel algorithms can behave extremely ineffectively, see Pingali et al. (2011). Different merits are derived for characterizing parallel systems, from simple speedup to price per performance Karp and Flatt (1990).

Because all of these difficulties, demands and possibilities, uncountable developments are running and planned to re-implement some existing code, written having a single processor in mind, for the already ubiquitous multi-core processors. To find out whether it is worth to invest in such efforts for parallelization, as well as to decide where to stop the development, one needs some quantitative measure of the parallelism. Even today, in the "multicore era", see Hill and Marty (2008), the performance is described by (a modified version of) Amdahl's law, see Amdahl, G. M. (1967). Unfortunately, Amdahl's law provides no support for the goals mentioned: it needs information on code architecture, makes assumptions which are not any more valid for modern accelerated processors and applications heavily using operating system services. However, scrutinizing the conditions and reverting the Amdahl's formula, it is possible to derive such a figure of merit.

## 2. LIMITATIONS OF PARALLELIZATION

### 2.1. Amdahl's law

Amdahl's considerations focus on the fact that some parts ( $P_i$ ) of a code can be parallelized, some ( $S_i$ ) must remain sequential. He also mentioned that data housekeeping causes some overhead, which in his paper was estimated to be in the range 20% to 40%, and that *the nature of this overhead appears to be sequential*.

Although Amdahl just wanted to draw the attention to the limitations of the single-processor approach applied to large-scale computing, his followers

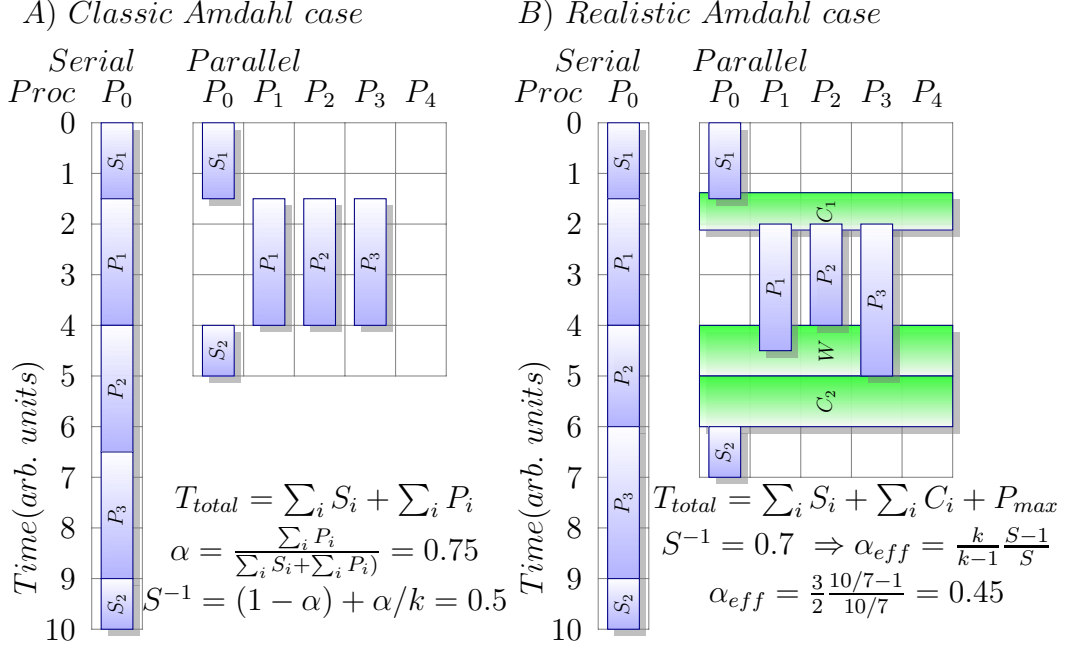


Figure 1: Illustrating Amdahl's law for idealistic and realistic cases

also provided a widely used formula, focussing on the parallelizable part of the code. As the left side of Fig 1 demonstrates, the usual interpretation implies three essential restrictions:

- the parallelized parts are of equal length in terms of execution time
- the housekeeping (controlling the parallelization, passing parameters, exchanging messages, etc.) has no costs in terms of execution time
- the number of parallelizable chunks coincides with the number of available computing resources

Essentially, this is why *Amdahl's law represents a theoretical upper limit for parallelization gain*. In Fig 1 the left side shows the idealistic case where the original process in the single-processor system comprises the sequential only parts  $S_i$ , and the parallelizable parts  $P_i$ . One can also see that the control components  $C_i$  are of the same nature as  $S_i$ , the non-parallelizable components. This also means that even in the idealistic case when  $S_i$  are negligible,  $C_i$  will represent a bound for parallelization. From the figure the meaning of  $\alpha$  is: in what fragment of time (in terms of the time needed for

completely sequential processing) the helper processors are utilized. When the task is scaled to several processors, the goal is obviously to maximize the utilization of the helper processors.

The realistic case (shown in the right side of Fig 1) however, is, that the parallelized parts are *not* of equal length (even if they contain exactly the same instructions, the hardware operation in modern processors may execute them in considerably different times (for examples see the operation of hardware accelerators inside a core or the network operation between processors, etc.) and also that the time required to control parallelization is not negligible and varying. Here  $\alpha_{eff}$  provides a value for an *average utilization* of the helper cores. Obviously, the unused cores and the unbalanced load of cores degrades this average utilization. To characterize the effects like sharing the processing between different number of helper cores (*the performance of the scaling*) or using different hardware conditions, one needs a quantitative figure of merit.

The figure also calls the attention to the fact that the static correspondence between program chunks and processing units can be very inefficient: all assigned processing units must wait for the delayed unit and also the capacity is lost if the number of computing resources exceeds the number of the parallelized chunks.

## 2.2. Factors affecting parallelism

Usually, Amdahl's law is expressed with the formula

$$S^{-1} = (1 - \alpha) + \alpha/k \quad (1)$$

where  $k$  is the number of parallelized code fragments,  $\alpha$  is the ratio of the parallelizable part to the total sequential part,  $S$  is the measurable speedup. The assumption can be visualized that (assuming many processing units) in  $\alpha$  fraction of the running time the processors are processing, in  $(1-\alpha)$  fraction they are waiting. I.e.  $\alpha$  describes how much, in average, the processors are utilized. Having those data, the resulting speedup can be estimated.

For a system under test, where  $\alpha$  is not *a priory* known, one can derive from the measurable speedup  $S$  an *effective parallelization* factor as

$$\alpha_{eff} = \frac{k}{k-1} \frac{S-1}{S} \quad (2)$$

where  $S$  is now the measured speedup, and  $k$  is the number of the available cores. Obviously, for the classical case,  $\alpha = \alpha_{eff}$ ; which simple means that

in *idealistic* case the actually measurable effective parallelization reaches the theoretically possible one. In other words,  $\alpha$  describes a system the *architecture* of which is completely known,  $\alpha_{eff}$  describes a system the *performance* of which is known from experiments. Again in other words,  $\alpha$  is the *theoretical upper limit*, which can hardly be reached, while  $\alpha_{eff}$  is the *experimental actual value*, which describes the complex architecture and the actual conditions. It is interesting to note, that  $\alpha_{eff}$  is an absolute measure of utilizing the available processing capacity, see section 3. Numerically  $(1 - \alpha_{eff})$  equals with the  $f$  value, established theoretically by Karp and Flatt (1990).

The  $\alpha_{eff}$  can then be used to refer back to the Amdahl's classical assumption even in the realistic case when the parallelized chunks have different length and the overhead to organize parallelization is not negligible. Note that in case of real tasks a kind of Sequential/Parallel Execution Model, see Yavits et al. (2014), shall be applied, which cannot use the simple picture reflected by  $\alpha$ , but  $\alpha_{eff}$  gives a good merit of the degree of parallelization for the duration of the execution of the process, and can be compared to the results of the technology-dependent parametrized formulas.

With our notations, in the classical Amdahl case on the left side in Fig. 1

$$S = \frac{\sum_i S_i + \sum_i P_i}{\sum_i S_i + \max_i P_i} = 2 \quad (3)$$

and

$$\alpha = \alpha_{eff} = \frac{\sum_i P_i}{\sum_i S_i + \sum_i P_i} = 3/4 \quad (4)$$

Now we can compare the effective parallelization in the two cases shown in Fig. 1. In the realistical case  $S = 10/7$ , which results in

$$\alpha_{eff} = \frac{3 \cdot 10/7 - 1}{2 - 10/7} = 0.45 \quad (5)$$

As seen, the overhead and the different duration of the parallelized parts reduced the effective parallelization drastically relative to the theoretically reachable value. Fig 2 gives a feeling on the effect of the computer system behaviour on the effective parallelization. The middle region (marked by balls) is mentioned by Amdahl as typical range of overhead. The asterisk in the figure shows the "working point" corresponding to the values used in Fig 1.

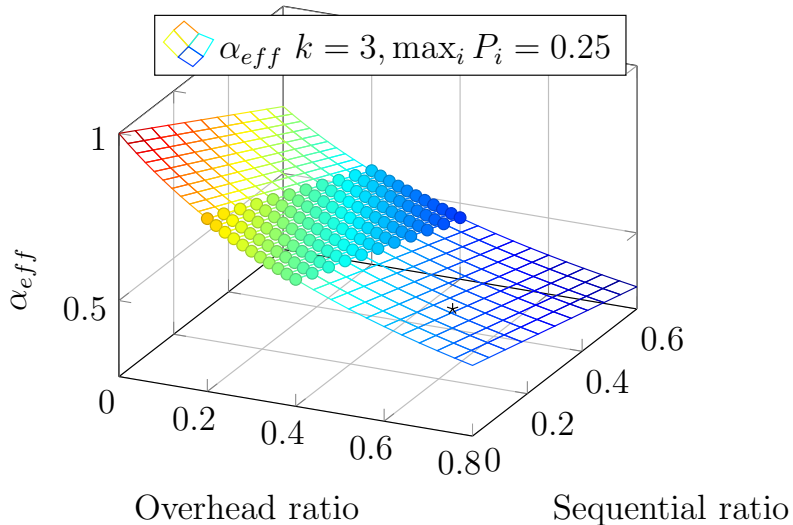


Figure 2: Behavior of the effective parallelization  $\alpha_{eff}$  in function of the overhead ratio (relative to the parallelizable payload execution length) and the ratio of the sequential part (relative to the total sequential execution time).

One can see that the effective parallelization drops quickly with both increasing overhead and sequential parts of the program. This fact draws the attention to the idea that *through decreasing either the control time or the sequential-only fraction of the code (or both), and utilizing the otherwise wasted processing capacity, a serious gain in the effective parallelization can be reached*. This was experienced in the "dynamic" architecture by Hill and Marty (2008).

### 2.3. Different implementations of parallelism

The timing analysis in Fig 1 can be applied to different kinds of parallelizations, from the processor-level parallelization (instruction or data level parallelization, in the nanoseconds range) to OS-level parallelization (including thread-level parallelization using several processors or cores, in the microseconds range), to network-level (between networked computers, like grids, in the milliseconds range). The principles are the same, see David et al. (2013), independently of the kind of implementation. In agreement with Yavits et al. (2014), housekeeping overhead is always present (and mainly

depends on the architectural solution), and remains a key question, although the main focus is always on reducing its effect.

The actual speedup (or the effective parallelization) depends strongly on the 'tricks' used during implementation. Although HW and SW parallelism are interpreted differently, they even can be combined, see Chandy and Singaraju (1999), resulting in hybrid architectures. For those greatly different architectural solutions it is even more hard to interpret  $\alpha$ , while  $\alpha_{eff}$  allows to compare different implementations (or the same implementation under different conditions) in such cases, too.

Notice that in all kinds of parallelization the relative overhead fulfills the observation made by Amdahl: for better performability the overhead time cannot exceed dozens of percents relative to the execution time of the parallelized chunk. Between networked computers the control times  $C_i$  are in the millisecond range and a lot of data must be transmitted, so for making parallelization efficient, typically complete jobs (having duration sometimes even in the minutes range) are sent to the other computers. When using thread-level parallelism through OS services inside the computer, the "expenses" of organizing threads is in the order of thousands of instructions, and typically the length of the working threads is also at least in that order or above. In hardware level parallelization, when using hyperthreading, the values of  $C_i$  are in the range of 1 clock cycle. However, the program chunks  $P_k$  are also in the order of a few clock cycles, i.e. in a comparable range. Similar holds for the speculative evaluation, the out-of-order evaluation, etc.

### 3. Practical applications

The good metric to select describing parallelism depends on many factors, see Karp and Flatt (1990). The newly introduced metric  $\alpha_{eff}$  describes how effectively the computing task is distributed between the processing units. As outlined above, the control functionality as well as inequalities in cutting program into equally long pieces (including data transfer between processing units) degrade  $\alpha_{eff}$ . Since  $1 - \alpha$  gives the sequential-only part of the program,  $(1 - \alpha_{eff})$  is expected to describe the ratio of the total (even unintended) sequential part, i.e. it is a sensitive measure of disturbances of parallelization. Since a larger load imbalance results in a larger decrease in value of  $\alpha_{eff}$  (as can be concluded from Karp and Flatt (1990)),  $\alpha_{eff}$  is a kind of derivative of relative speedup), problems can be identified better than from speedup values. If the parallelization is well-organized (load balanced, small overhead,

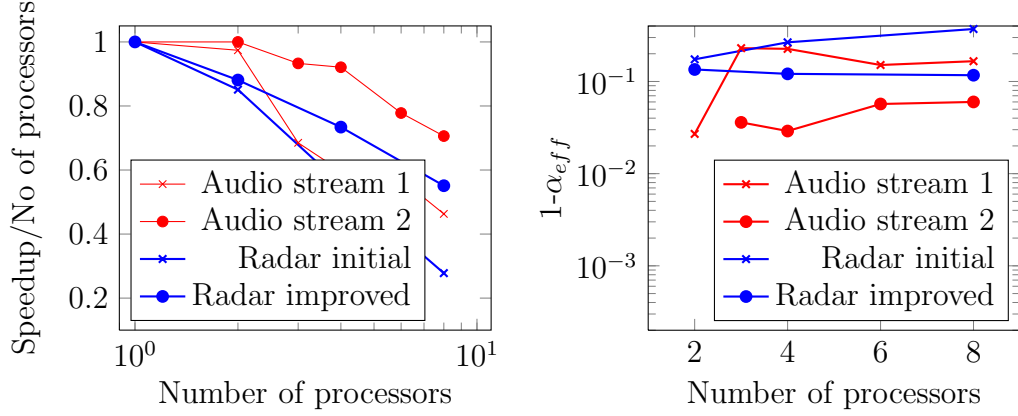


Figure 3: Relative speedup (left side) and  $(1 - \alpha_{eff})$  (right side) values, measured running the audio and radar processing on different number of cores. Sheng et al. (2014)

right number of processors),  $\alpha_{eff}$  saturates at unity, so tendencies can be better displayed through using  $(1 - \alpha_{eff})$ .

### 3.1. Characterizing parallelization efforts

In paper Sheng et al. (2014) a compiler making effective parallelization of an originally sequential code for different number of cores is described and validated by running the executable code on platforms having the corresponding number of cores. Let us apply Equ. (2) to their results, shown in Figs 8 and 10 in their paper.

Fig. 3 left side displays efficiency (Efficiency = speedup divided by the number of cores) in function of number of cores for two different processings of audio streams, and for two processings of radar signals. The data displayed in the figures are derived simply through reading back diagram values from the mentioned figures in Sheng et al. (2014), so they may be not accurate. However, they are accurate enough to support our conclusions.

Based on their merit, the authors of Sheng et al. (2014) can only declare a qualitative statement, that the 'efficiency' decreases less steadily (dots on the figure) with the growing number of cores "The higher number of parallel processes in Audio-2 gives better results", if they consider load balancing.

It can surely be stated, that the improvement was successful: in both cases the decrease with increasing number of cores is less steep. The diagrams cannot tell, however, whether further improvements are possible or



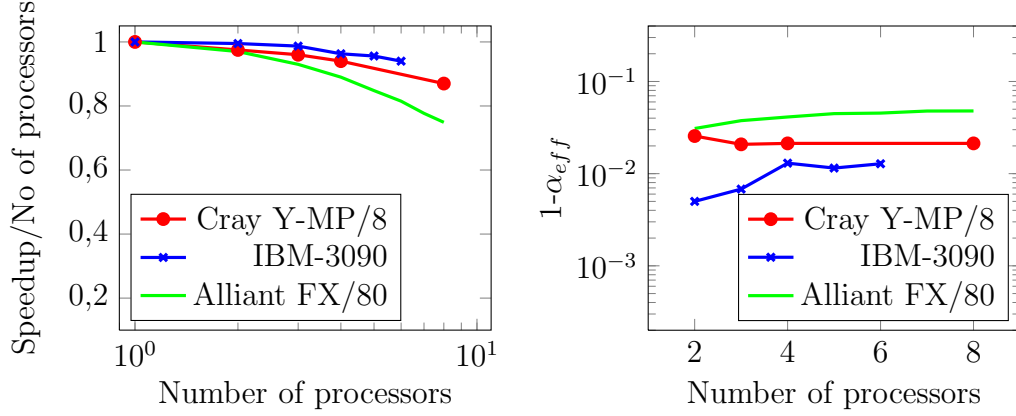


Figure 4: Relative speedup (left side) and  $(1 - \alpha_{eff})$  (right side) values, measured running Linpack on different computers with different number of parallel processors. Karp and Flatt (1990)

whether the parallelization is uniform in function of the number of cores. In contrast, the  $(1 - \alpha_{eff})$  diagrams (right side) show also, that in both cases the improvement decreased the sequential part, i.e. improved the parallelization. It can also be seen, that in the case of audio stream, the parallelization is improved and so did the uniformity of parallelization. In the case of radar signals, without optimization the parallelization decreases as the number of the cores increases. With load balancing option on, the parallelization is at any core number gets better. The compiler really does a good job:  $\alpha_{eff}$  is practically constant, the compiler finds nearly all possibilities:

Note that the absolute values in the two cases must not be compared: they represent the sequential-only part of the two programs, and they might be different for the different programs. The uniformity of the values make also highly probable, that in the case of audio streams further optimization can be done, at least for the 2-core and 3-core systems, while processing of radar signals reached its bounds. In addition, it can also be estimated, that the non-parallelizable part amounts to  $\approx < 10\%$ .

Notice that using  $S$  and  $\alpha_{eff}$  are simply two different points of view of the same thing. If we have the information, how big is the  $\alpha$  fraction of the code which can be executed in parallel (an architectural point of view), we can estimate the maximum speedup we can reach. Here we assume that all processors have the same architecture. The experimentalist's point of view

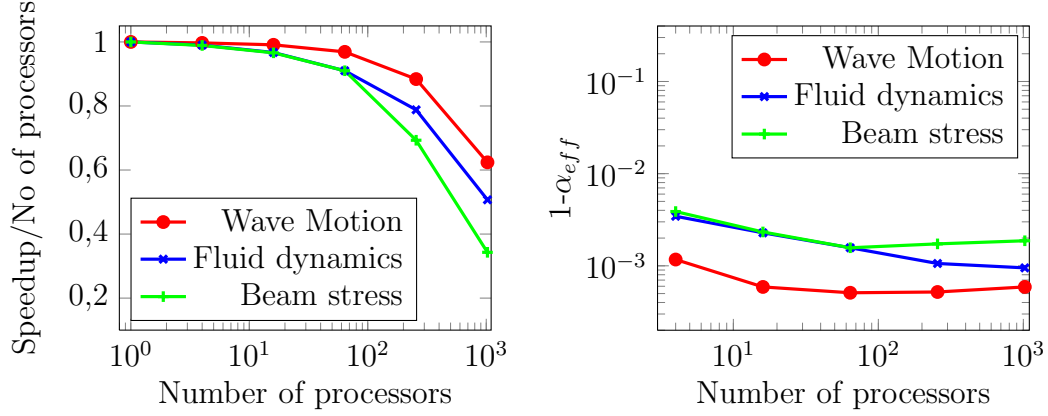


Figure 5: Relative speedup (left side) and  $(1 - \alpha_{eff})$  (right side) values, measured running different algorithms on the same computer with different number of parallel processors. Karp and Flatt (1990)

is different: if we can measure the speedup, and know how many processing units was used, we can estimate how big  $\alpha_{eff}$  fraction was running in parallel (assuming the mentioned ideal conditions). Notice also that in deriving  $\alpha_{eff}$  no assumption was made on the code architecture or the nature of the computing units or their way of linking, so the merit can be used to characterize the effect of the *change*, if one of the mentioned components changes. It means, one can use that merit for describing the effect of changing the code architecture, or the (behavior of the) interconnecting network, the (internal architecture of the) hardware setup, etc. as well.

### 3.2. Characterizing HW architectures

In Table I of Karp and Flatt (1990), different architectures are compared, running the same program (Linpack) on computers from different manufacturers and having different number of processors. Because the subject of the paper was deriving a metric from measured data, here the precision of the values is much better. The high degree of parallelization results in  $\alpha_{eff}$  values, close to unity, so the value  $(1 - \alpha_{eff})$  is used in Figure 4.

As also in the previous case, the efficiency decreases with the increasing number of cores. The effective parallelization is nearly constant, and the difference in the absolute values can be attributed to implementation details of the different computers.

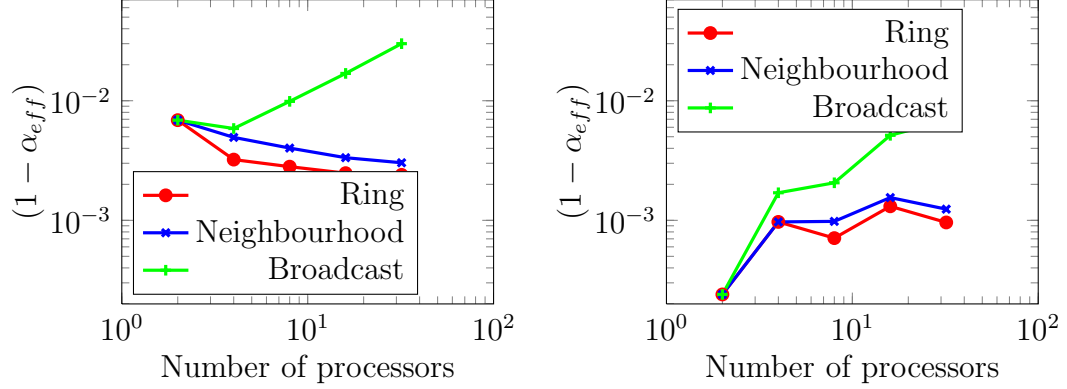


Figure 6:  $(1 - \alpha_{eff})$  values, measured when minimizing Rosenbrock function (left side) and Rastrigin function (right side), on the same SoC, using different communication strategies, in function of the used processors. de Macedo Mourelle et al. (2016)

### 3.3. Comparing communication strategies in Systems-on-Chip

In their work de Macedo Mourelle et al. (2016) the authors compare different communication strategies their PSO uses when minimizing the Rosenbrock function and the Rastrigin function, respectively. As it could be expected, in the case of the 'broadcast' type communication the 'sequential' fraction increases with the number of the processors, in other cases practically remains constant. The fluctuation shows the limitations of the (otherwise excellent) measurement precision.

It is worth to compare Fig. 6 with Fig 5, right side. All the three diagrams show the scaling behavior of some procedures, in function of the processing units. It would be worth to run the processes shown in Fig 5, in the PSO, to find out the advantage of having the processing units *inside* the chip.

### 3.4. Characterizing scaling of parallelization

In Table II of Karp and Flatt (1990), execution time of different programs are given in function of processors. The data are shown in Fig. 5. As presented there, the efficiency drops in a catastrophic way as the number of cores increases, while  $(1 - \alpha_{eff})$  changes only within the limits of the measurement error. Notice that Figs. 3, 4 and 5 use the same scale, and that the steeper decrease of efficiency means higher values of  $(1 - \alpha_{eff})$ .

The behavior of efficiency deserves some analysis. As detailed at the beginning of section 2.1, the distinguished constituent in Amdahl's classic analysis is the parallelizable fraction  $\alpha$ , all the rest (including wait time, non-payload activity, etc) goes into the "sequential-only" fraction. When using several processors, one of them makes the sequential calculation, the others are waiting (use the same amount of time). So, when calculating the speedup, one calculates

$$S = \frac{(1 - \alpha) + \alpha}{(1 - \alpha) + \alpha/k} = \frac{k}{k(1 - \alpha) + \alpha} \quad (6)$$

hence the efficiency

$$\frac{S}{k} = \frac{1}{k(1 - \alpha) + \alpha} \quad (7)$$

This explains the behavior of diagram  $\frac{S}{k}$  in function of  $k$  in figures above: the more processors, the lower efficiency. In the case of Fig. 5,  $(1 - \alpha)$  is in the order of  $10^{-3}$ , so the efficiency decreases to 0.5 at  $10^3$  processors, while in the case of Fig. 3,  $(1 - \alpha)$  is  $\approx 10^{-1}$ , so the efficiency decreases to 0.5 at  $10^1$  processors. This is why Amdahl made his very reasonable conclusion: "the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude" Amdahl, G. M. (1967).

#### 4. Conclusions

With the spread of both multi-core architectures, using different parallelization solutions (like different networking or reconfigurable connection of cores, etc.) and parallelizing the formerly sequential code either with programmers' effort or using parallelizing compilers like the one by Sheng et al. (2014), it becomes more and more important problem to characterize quantitatively the performance of the parallelization. Through inverting the formula known as Amdahl's law, and re-interpreting the comprised quantities, such a figure of merit was derived. This experimental quantity correctly describes the performance of parallelization, allowing to characterize the performance of programmers or parallelizing compilers (see Fig. 3), different architectural solutions with many processors (see Fig. 4), different algorithms in function of the number of the processors (see Fig. 5), as well as describing the performance of the network connection during running the task, or

quantifying the synchronization method used between the computing units. The introduced merit seems to be an adequate measure of the performance of the technology used for parallelization, unlike the formerly used quantity (speedup divided by the number of computing units).

## References

- Agarwal, V., Hrishikesh, M., Keckler, S., Burger, D., 2000. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In: Proceedings of the 27th Annual International Symposium on Computer Architecture.
- Amdahl, G. M., 1967. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In: AFIPS Conference Proceedings. Vol. 30. pp. 483–485.
- Chandy, J. A., Singaraju, J., 1999. Hardware parallelism vs. software parallelism. In: USENIX Summer Conference.  
URL [http://static.usenix.org/event/hotpar09/tech/full\\_papers/chandy/chandy\\_html](http://static.usenix.org/event/hotpar09/tech/full_papers/chandy/chandy_html)
- David, T., Guerraoui, R., Trigonakis, V., 2013. Everything you always wanted to know about synchronization but were afraid to ask. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). pp. 33–48.
- de Macedo Mourelle, L., Nedjah, N., Pessanha, F. G., 2016. Reconfigurable and Adaptive Computing: Theory and Applications. CRC press, Ch. Chapter 5: Interprocess Communication via Crossbar for Shared Memory Systems-on-chip.
- Fuller, S. H., Millett, L. I., 2011. Computing Performance: Game Over or Next Level? Computer 44, 31–38, [http://download.nap.edu/cart/download.cgi?&record\\_id=12980](http://download.nap.edu/cart/download.cgi?&record_id=12980).
- Hill, M. D., Marty, M. R., 2008. Amdahls Law in the Multicore Era. IEEE Computer 41 (7), 33–38.
- Karp, A. H., Flatt, H. P., May 1990. Measuring parallel processor performance. Commun. ACM 33 (5), 539–543.  
URL <http://doi.acm.org/10.1145/78607.78614>

- Pingali, K., Nguyen, D., Kulkarni, M., Burtscher, M., Hassaan, M. A., Kaleem, R., Lee, T.-H., Lenharth, A., Manevich, R., Méndez-Lojo, M., Prountzos, D., Sui, X., Jun. 2011. The Tao of Parallelism in Algorithms. SIGPLAN Not. 46 (6), 12–25.  
URL <http://doi.acm.org/10.1145/1993316.1993501>
- Sheng, W., Schürmans, S., Odendahl, M., Bertsch, M., Volevach, V., Leupers, R., Ascheid, G., 2014. A compiler infrastructure for embedded heterogeneous MPSoCs. *Parallel Computing* 40, 51–68.
- S(o)OS project, 2010. Resource-independent execution support on exa-scale systems. <http://www.soos-project.eu/index.php/related-initiatives>.
- Yang, J., Cui, H., Wu, J., Tang, Y., Hu, G., 2014. Making Parallel Programs Reliable with Stable Multithreading. *Communications of the ACM* 57 (3), 58–69.
- Yavits, L., Morad, A., Ginosar, R., 2014. The effect of communication and synchronization on Amdahls law in multicore systems. *Parallel Computing* 40 (1), 1–16.